

**CS 580 Client-Server Programming**  
**Spring Semester, 2005**  
**Doc 19 Databases & Architecture**  
**Contents**

Databases & Architecture.....	2
Organizing Domain Logic.....	6
Transaction Script.....	7
Domain Model.....	8
Table Module.....	9
Organizing Access to Database.....	10
Table Data Gateway.....	11
Row Data Gateway.....	14
Active Record.....	16
Object-Relational Mapping Layers.....	20
Hibernate Simple Example.....	21
Object Databases.....	26

**References**

Patterns of Enterprise Application Architecture, Martin Folwer,  
Addison-Wesley, 2003

CS 683 Lecture document 27 Hibernate Example, Fall 2005,  
[http://www.eli.sdsu.edu/courses/fall04/cs683/notes/hibernate/  
hibernate.html](http://www.eli.sdsu.edu/courses/fall04/cs683/notes/hibernate/hibernate.html)

Copyright ©, All rights reserved. 2005 SDSU & Roger Whitney, 5500 Campanile Drive, San Diego, CA 92182-7700 USA. OpenContent (<http://www.opencontent.org/opl.shtml>) license defines the copyright on this document.

## **Databases & Architecture**

How to keep SQL isolated?

How to isolate database connection details?

How to keep dealing with the database under control?

How to structure programs that use databases?

## Example – Office Hours

### Common Operations

- Find Office hours for instructor X
- Find office hours of any graduate advisor
- Find office hours of any undergraduate advisor
- Find office hours of any TA
- Who has office hours at time X
- What times are there no office hours
- Add office hours
- Modify office hours

### Tables

Faculty			
Id	Name	Office	Phone
1	Eckberg	GMCS-543	594-6834
2	Donald	GMCS-541	594-7248
3	Carroll	GMCS-537	594-7242

OfficeHours				
Id	StartTime	EndTime	Day	FacultyId
1	10:00	11:00	Tuesday	1
2	10:00	11:00	Thursday	1

RoleTypes	
ID	Role
1	Undergraduate Advisor
2	Graduate Advisor
3	TA

Roles	
FacultyId	TypeId
1	2
2	2
3	1

## DatabaseConnector

Hides username and password

Can we hide the connections completely?

Should we hide connections?

```
public class DatabaseConnector {
    private String databaseUrl;
    private String user;
    private String password;
    private ArrayList connectionPool;

    private static DatabaseConnector instance =
        DatabaseConnector("filename");

    public static DatabaseConnector instance() {
        return instance;
    }

    private DatabaseConnector(String filename) {
        read file for database info
        set private fields
    }

    public ResultSet executeQuery( String sql ) {
        return getStatement().executeQuery( sql);
    }

    public Statement getStatement() {
        return getConnection().createStatement();
    }

    private Connection getConnection() { return a connection}

    etc
}
```

## Organizing Domain Logic

How to organize an application that uses a database

Fowler provides the following methods

- Transaction Script
- Domain Model
- Table Module
- Service Layer

## Transaction Script

Identify different transactions to be performed by the application

Each transaction is handled by a separate method

## Consequences

- Very simple to implement
- As application grows in complexity, becomes overly complex and hard to manage

```
public class TorrentData {  
    public Metadata getFile(String id) {  
        code & SQL to get file from database }  
  
    public byte[] getPiece(String fileId, int pieceIndex) {  
        code & SQL to get piece from database}  
  
    public void setPiece(String fileId, int pieceIndex,  
        byte[] peice) {  
        code & SQL to put piece into database}  
  
    etc.
```

## **Domain Model**

Implement classes that incorporates both behavior & data

Classes represent objects in the domain

Program becomes collection of interacting objects

Objects map to tables

- A single object may span many tables
- A table row may contain multiple objects

## **Consequences**

Overly complex for simple applications

Scales well to complex applications

Database organizes data differently

## **Table Module**

For each table (or view) implement a class

Each class holds the business logic related to the data in the table

## **Consequences**

Classes are organized around database structure rather than OO principles

Handles more complex situations than Transaction Script

Not as scalable as Domain Model

## Organizing Access to Database

- Table Data Gateway
- Row Data Gateway
- Active Record
- Data Mapper

## Table Data Gateway

One object handles all the rows in a table or view

Each table has one class that knows the table

One object represents the table – all the rows

Gateway hides all the Sql from the rest of the program

Works well with

- Table Module
- Transaction Script

## OfficeHoursGateway

```
public class OfficeHoursGateway {

    private static String addOfficeHoursSql =
        "INSERT
        INTO officeHours ( startTime, endTime, day, facultyId )
        VALUES ( ? , ? , '?' , ?)";

    Private static String officeHoursSql =
        "SELECT startTime, endTime, day
        FROM officeHours
        WHERE facultyId = ?";

    public ResultSet officeHoursFor(int facultyId,) {
        Statement hoursStatement = DatabaseConnector.instance().
            prepareStatement(officeHoursSql);
        hoursStatement.setObject( 1, facultyId);
        return hoursStatement.executeQuery();
    }

    public int setOfficeHoursFor(int facultyId, Time start, Time end,
String day) {

        Statement addOfficeHours = DatabaseConnector.instance().
            prepareStatement(addOfficeHoursSql);

        addOfficeHours.setObject(1, start);
        addOfficeHours.setObject(2, end);
        addOfficeHours.setObject(3, day);
        addOfficeHours.setObject(4, facultyId);
        return addOfficeHours.executeQuery();
    }
}
```

## Transaction Script + Table Gateway

```
public class OfficeHoursServer {
    private OfficeHoursGateway officeHours;
    private FacultyGateway faculty;
    etc.

    public Vector officeHoursFor(String facultyName) {

        int facultyId = faculty.idFor(facultyName,);

        ResultSet officeHoursRows = officeHours.officeHoursFor(
facultyId);
        Vector officeHours = new Vector();
        while (officeHoursRows.next() ) {
            Dictionary officeHour = new Dictionary();
            officeHour.put( "start", officeHoursRows.getObject( "start"));
            officeHour.put( "end", officeHoursRows.getObject( "end"));
            officeHour.put( "day", officeHoursRows.getObject( "day"));
            officeHours.add( officeHour);
        }
        officeHoursRows.close();
        return officeHours;
    }

    etc.
}
```

## **Row Data Gateway**

One object handles or represents a single row in a table or view

Each table has one class that knows the table

Gateway hides all the Sql from the rest of the program

A class provides just accessor methods to data in a row

Works well with Transaction script

## [sdsu.sql.DatabaseTable](#) Utility for Row Access

Part of [SDSU Java library](#)

Some Creation methods

```
Connection db;  
db = DriverManager.getConnection( dbUrl, user, password);  
  
DatabaseTable rows;  
  
//Get rows from table Faculty with column Name = Donald  
rows = DatabaseTable.getRow("Faculty", "Name", "Donald", db);  
rows.elementAt(rowIndex, "Office");  
  
// Get rows returned from a SQL select statment  
rows = DatabaseTable.fromSQL("a SQL select", db);
```

## Active Record

Each domain object know how add/remove/find it state in the database

In simple cases

- Class for each table
- An object represents one row in the table
- Similar to Row Data Gateway with domain logic

## Faculty

```
public class Faculty {
    String name;
    String phoneNumber;
    int id;
    etc.

    private final static String findByNameSql =
        "SELECT *
        FROM faculty
        WHERE name = '?'";

    public static Faculty findByName(String name ) {
        Statement find =
            databaseConnector.prepareStatement(findByNameSql);
        find.setObject( 1, name);
        ResultSet facultyRow = find.executeQuery();
        return load(facultyRow);
    }

    public static Faculty load( ResultSet facultyRow) {
        create faculty object.
        get data out of Resultset.
        Put data into faculty object.
        Return faculty object.
    }
}
```

```
public boolean hasOfficeHoursAt(Time anHour) {
    Iterator hours = officeHours().iterator();
    while (hours.hasNext() ) {
        OfficeHour officeHour = (OfficeHour) hours.next();
        if (officeHour.contains( anHour) ) return true;
    }
    return false;
}

public ArrayList officeHours() {
    if( officeHours = nil ) {
        officeHours = OfficeHour.findFor( id );
    }
    return officeHours;
}
```

## Domain Model + Active Record

```
public class OfficeHoursServer {  
    public Vector officeHoursFor(String facultyName) {  
        Faculty X = Faculty.findByName (facultyName,);  
        ArrayList officeHours = X.officeHours();  
        Convert contents of officeHOurs to XML-RPC acceptable types  
        return vector of valid XML-RPC types;  
    }  
    etc.  
}
```

## Object-Relational Mapping Layers

Implementing a good object-relational layer is a lot of work

Use existing tools to save a lot of time

Read/Write objects from tables without SQL

Some existing object-relational layers

- [JDO](#) – Java Data Object (Java framework)
- [TopLink](#) (Commercial - Java)
- [Hibernate](#) (Open source - Java)
- [Cayenne](#) (Open source - Java)
- [GLORP](#) (Open source - Smalltalk)

## Hibernate Simple Example

Storing Person objects in table

### Database Table

id	first_name	last_name

### SQL Used to Create Table

```
CREATE TABLE PEOPLE  
  (FIRST_NAME varchar(50) NULL ,  
   LAST_NAME varchar(50) NULL ,  
   ID int NOT NULL ,  
   PRIMARY KEY (id));
```

## People Class

```
package sample;

public class People {
    String firstName;
    String lastName;
    long id;

    public People () {super(); }

    public People(String first, String last) {
        firstName = first;
        lastName = last;
    }

    public String getLastName() { return lastName; }

    public String getFirstName() { return firstName; }

    public void setFirstName( String name) { firstName = name; }

    public void setLastName( String name) { lastName = name; }

    public long getId() { return id; }

    public void setId(long l) {id = l; }

    public String toString() {return firstName + " " + lastName + id;}
}
```

## Mapping – People.hbm.xml

Indicates how to map object fields to table columns

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
  "-//Hibernate/Hibernate Mapping DTD//EN"
  "http://hibernate.sourceforge.net/hibernate-mapping-2.0.dtd" >
<hibernate-mapping package="sample">
  <class
    name="People"
    table="people" >
    <id
      name="id"
      type="java.lang.Long"
      column="id" >
      <generator class="assigned"/>
    </id>
    <property
      name="lastName"
      column="last_name"
      type="string"
      not-null="false"
      length="50" />
    <property
      name="firstName"
      column="first_name"
      type="string"
      not-null="false"
      length="50" />
  </class>
</hibernate-mapping>
```

## Sample Connection

```
import lots of stuff;

public class Main {
    public static void main(String[] args) throws Exception {
        sampleRead();
        sampleWrite();
    }

    static Session getHibernateSession() throws
        MappingException, HibernateException, Exception {
        some code to get HibernateSession
    }

    static void sampleWrite() throws
        MappingException, HibernateException, Exception {
        Session session = getHibernateSession();
        Transaction save = session.beginTransaction();
        People newPerson = new People("Jack", "Frost");
        newPerson.setId(1);
        session.save(newPerson);
        newPerson = new People("Jack", "Ripper");
        newPerson.setId(2);
        session.save(newPerson);
        save.commit();
        session.close();
    }
}
```

## Sample Connection Continued

```
static void sampleRead() throws
    MappingException, HibernateException, Exception {
    Session session = getHibernateSession();
    Query getByLastName =
        session.createQuery(
            "from People p where p.lastName = :var");
    getByLastName.setString("var", "Frost");
    List result = getByLastName.list();
    System.out.println("Number of Objects: " + result.size());
    People frost = (People) result.get( 0);
    System.out.println(frost);
    session.close();
}
}
```

## Object Databases

- Stores objects without tables
- Stores references to objects

A partial list

- Gemstone (<http://www.gemstone.com/>)
- Objectivity (<http://www.objectivity.com/>)
- Matisse (<http://www.matisse.com/>)
- OmniBase (<http://www.gorisek.com/homepage/index.html>)
- Versant (<http://www.versant.com/index>)
- ObjectStore (<http://www.objectstore.net/index.ssp>)
- Zope Object Database (<http://zope.org/Wikis/ZODB/FrontPage>)

No need to convert between objects and sql

## OmniBase Example

### Create the Database

```
database :=OmniBase createOn: 'bittorrent'.  
  
[OmniBase root  
  at: 'files'  
  put: Set newPersistent  
] evaluateAndCommitIn: database newTransaction.
```

OmniBase root is a Dictionary

Entry point to data

### Adding a new file

```
Server>>add: aMetaFile  
  
[types := OmniBase root at: 'files'.  
types add: aMetaFile.  
types markDirty] evaluateAndCommitIn: database newTransaction.
```

### Finding files by name

```
Server>>filesWithName: aString  
  
[^ (OmniBase root at: 'files')  
  select: [:each | aString match: each name ]  
  evaluateIn: database newTransaction.
```